2021

# Scripting Unix



## Commandes utiles

Olivier D.

# Table des matières

# 1 Introduction

**Idempotence**

L'idempotence est une notion primordiale en scripting. Cela signifie qu'une opération a le même effet qu'on l'applique une ou plusieurs fois.

`apt` est idempotent

`mkdir –p dossier` est idempotent

`rm –f fichier` est idempotent

`if ! grep –qs "root = /" /etc/wsl.conf ; then echo "root = /" >> /etc/wsl.conf ; fi` est idempotent

`grep –q` : n'affiche rien si la ligne est trouvé et `$? = 0`

`grep –s` : n'affiche rien si le fichier n'est pas trouvé et `$? = 1`

`if sans crochets` teste si le résultat est égal à 0

`!` : ce caractère vérifie si le résultat n'est pas zéro : si c'est vrai alors le résultat de

**Exemple :**

```
wsl_config_file="/etc/wsl.conf"
config_line="root = /"
if [[ –f $wsl_config_file ]] ; then                          # test si le fichier existe
  if ! grep "$config_line" "$wsl_config_file" ; then         # test si la ligne n'existe pas
    echo "$config_line" >> "$wsl_config_file"
  fi
fi
```

nota : `[[ –f fichier ]]` : teste si un fichier existe

**Paramétrage de vi / vim**

`vi ~/.exrc` : fichier des paramètres de `vi`

```
set numbers                     # afficher les numéros de lignes
set ignorecase                  # ignore la casse
set syntax=on                   # couleurs de syntaxe
set nocompatible
set autoindent                  # met en place l'auto-indentation
```

# 2 Notions de bases

**Le « she bang »**

`#!/bin/bash` : à toujours placer en début de script, indique le Shell utilisé. Sinon le script s'exécute en Shell utilisateur
`#!/bin/sh` : que le Bourne Shell pour Unix

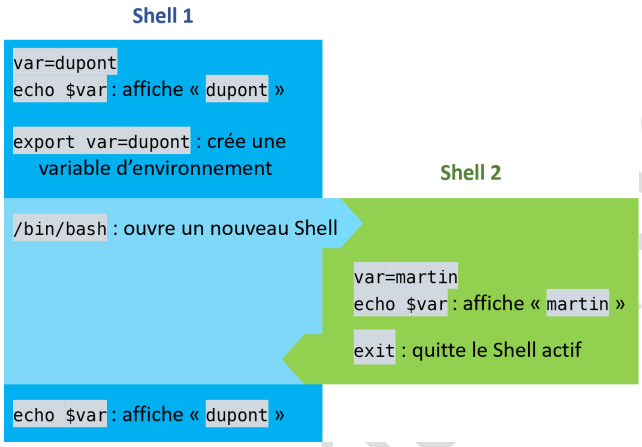
**Variable VS Variable d'environnement**

Ecrire une variable : `${variable}`

Afficher le résultat d'une variable : `echo $(ls directory)` (au lieu d'utiliser `echo `ls directory``)

`set` : liste de toutes les variables qui sont présentes uniquement dans le Shell en cours

`env` : liste des variables d'environnement qui sont présent dans tous les Shell et sous-Shell




*Les variables d'environnements survivent aux variables du Shell*

Nota : une variable ne peut pas commencer par `[0-9]` et ne contient que `[a-Z] + [_] + [0-9]`

`echo $$` : numéro de processus courant

`echo $?` : résultat de la commande précédente (`0` = la commande s'est bien passée ; autre = erreur)

`$1` - `$9` : argument entré à la suite de la commande d'exécution du script

- Exemple : `./test.sh olivier dehecq`
- `echo $1` renvoie `olivier`
- `echo $2` renvoie dehecq)
- `echo $0` renvoie `./test.sh` : script appelé)
- `echo $@` renvoie tous les arguments du script

`echo $#` : nombre d'arguments envoyés (2 dans la commande précédente)

`'...'` (simple quote) : isole tous les caractères.

`"..."` (double quote) : isole tous les caractères sauf : `, \ et $

**Caractères spéciaux du shell**

`;` (exemple : `ls ; date`) : sert à enchainer deux commandes, exécutées l'une après l'autre

`&&` (exemple : `ls && date`) : la commande de droite sera exécutée que si la partie gauche s'est bien exécutée

- Exemple : `ls test.sh && echo OK`

`||` (exemple : `ls && date`) : la commande de droite ne sera exécutée que si la commande à gauche ne s'est pas bien exécutée. Exemple : `ls test.sh || echo KO`

**Regroupement des commandes**

`( ls /bin ; ls /usr/bin ) | wc –l` : ce qui est entre `( ... )` est exécuté dans un sous Shell

`{ ls /bin ; ls /usr/bin; } | wc –l` : ce qui est entre `{ ...; }` est exécuté dans un sous Shell

- Exemple : `ls xys || { echo "fichier introuvable" ; var = Alerte ; }`

**Utiliser les couleurs dans le Shell**

| Couleur | Caractère | Fond |
|---|---|---|
| Noir | 30 | 40 |
| Rouge | 31 | 41 |
| Vert | 32 | 42 |
| Brun | 33 | 43 |
| Bleu | 34 | 44 |
| Magenta | 35 | 45 |
| Cyan | 36 | 46 |
| Blanc | 37 | 47 |

| Attribut |
|---|
| 0 : aucun |
| 1 : **gras** |
| 4 : souligné |
| 7 : inversé |
| 8 : invisible |
| 9 : ~~barré~~ |
| |
| |

`\033[1;32m` : texte vert gras

- `\033[` : annonce de couleur
- `1;32` : arguments
- `m` : fin de la définition

Exemple : `echo "Bonjour sur \033[1;32mwww.informatique1.fr"`

```
Bonjour sur www.informatique1.fr
```

**Les redirections**

`a | b` : envoyer le output de `a` en input de `b`

`echo a > fichier` : écraser le `fichier` et le remplacer par le résultat de la commande `echo a`

`echo a >> fichier` : indente le `fichier` avec le résultat de la commande `echo a`

`a < fichier`: envoyer le contenu de `fichier` dans l'input de la commande `a` (équivalent à `cat fichier | a`)

`a << EOF` : exécute la fonction `a` à partir de l'output du terminal (ou de la suite) jusqu'à ce que le token `EOF` (peut avoir une autre valeur que OEF) soit trouvé dans l'output

Exemple : écrit le contenu jusqu'à EOF dans fichier.txt

```
variable = "info1"
cat << EOF > fichier.txt
[config]
valeur=123
username=$variable
EOF
```

Nota : il est possible d'utiliser une autre suite de caractère que `EOF`. Il faut juste que les deux suites soient identiques.

Cela fonctionnerait aussi si on remplaçait les deux occurrences de `EOF` par `TITITATATOTO`

**Les enchainements de commandes**

`a || b` : exécute `b` si la commande `a` échoue

`a && b` : exécute `b` si la commande `a` réussit

`a ; b` : exécute la commande `a` puis la commande `b`

`a &` : exécute la commande `a` puis la passe en arrière-plan

# 3 Les commandes internes au Shell

**read**

`read` : stocke une chaine entrée sous la variable voulue. Remplace PAUSE

`vi tel.sh`

```
echo "saisissez votre nom : "
read NOM
echo "saisissez votre numéro de téléphone : "
read TEL
echo "bonjour $NOM, votre numéro de tel est le : $TEL"
```

**echo**

`echo` : affichage de texte et de variables

`–e` : en Bash, interprète les caractères spéciaux (ex. : `echo –e "\nCoucou\n"`)

`\n` : nouvelle ligne

`\t` : tabulation

`\c` : enlève le retour à la ligne après

**exec**

`exec` : exécute une commande dans le Shell en cours puis le quitte

Exemple : `exec ls`

**exit**

`exit` : arrête le Shell courant et définit la valeur de `$?`

**expr**

`expr` calcule et renvoit le résultat d'un calcul arithmétique.

Exemple : `expr 12 + 8` renvoie 20. `expr 12 \* 2` renvoie 24

Préférer `let` et `(( ... ))`

# 4 Les conditions

/!\ attention aux espaces avant et après les `[ ... ]` et `[[ ... ]]` car `[[` et `]]` sont des fonctions.

**test**

page de man : test(1): check file types/compare values - Linux man page (die.net)

syntaxe : `test ...` ou `[ ... ]`

exemple : `[ -f xyz ] || echo KO`

**Test sur les fichiers :**

- `-f` : est un fichier
- `-d` : est un répertoire
- `-x` : le droit d'exécution est accordé sur le fichier
- `-s` : la taille du fichier est supérieure à zéro

**Opérateur de chaine**

- `[ "$NOM" = "root" ] && ...` : la variable $NOM est égale à « root »
- `!=` : différent de
- `-n` : variable non nulle
- `-z` : variable nulle. Exemple : `[ -z "$1" ]` : le premier argument est nul

**Test sur les valeurs numériques**

- `[ "$AGE" -eq 18 ]` : égal
- `-ne` : différent
- `-gt` : plus grand que. `-ge` : plus grand ou égal à
- `-lt` : plus petit que. `-le` : plus petit ou égal à

**Combinaisons**

- `-a` : et
- `-o` : ou
- `!` : négation de l'expression suivante
- `\( \)` : regroupement
- Exemple : `[ ! \( "a" = "$HOME" -o 3 -lt 4 \) ]; echo $?` : résultat inverse de $HOME= « a » ou 3 < 4

**Pour la comparaison de chaines**

`[[ ... ]]` : utiliser des doubles crochets pour faire des comparaisons

- `*` : 0 à n caractères
- `?` : 1 caractère
- `[abc]` : un caractère parmi a, b ou c
- `[!xyz]` : un caractère qui n'est ni x, ni y, ni z
- Exemple : `[[ $VAR == *Lin[aeiou]x* ]] && echo OK`

`shopt | grep extglob` : chercher si extglob est activé

```
extglob         on
```

`shopt -s extglob` : activer les caractères étendus dans les modèles de recherche). `-u` désactiver

- `?(...)` : 0 à 1 fois la chaine
- `*(...)` : 0 à n fois la chaine
- `+(...)` : 1 à n fois la chaine
- `@(...)` : 1 fois la chaine
- `!(...)` : 0 fois la chaine
- `*(...|...)` : de 0 à n fois la chaine1 **ou** la chaine2 (fonctionne avec `*+@!`)
- Exemple : `[[ var = ?([+-]+([0-9]) ]] && echo "Le nombre entier est valide"`

**if … else**

**if then fi :**

```
if [ -f /var/fichier.txt ]
then
  echo "le fichier fichier.txt est présent"
  echo "super nouvelle"
fi
```

**if then else fi :**

```
if [[ $sexe == "male" ]]
  then
  echo "you are a male"
else
  echo "you are a female"
  echo "or a curiosity"
fi
```

**if then else elif then fi :**

```
if [ $money -gt 1000000000 ]
  then
  echo "it seems that you have a lot of money"
  echo "or your money exchange rate is low"
elif [ $money -lt 100 ]
  then
  echo "it seems that you have little money"
  echo "or your exchange rate is high"
fi
```

**case**

permet de faire des actions selon la valeur attendue d'une variable.

```
case $COULEUR in
noir) echo "vous avez choisi noir" ;;
blanc) echo "c'est le banc qui est choisi" ;;
rouge) echo "là c'est le rouge" ;;
jaune|orange)  echo "orange ou jaune c'est du pareil au même" ;;
*) echo "la couleur choisie n'est pas autorisée" ;;
esac
```

**while**

syntaxe : `while [ condition ]; do commands; done`

Boucle tant que la condition est vraie.

```
while [ -z "$NOM" ]
  do
  read -p "Entrez votre nom : " NOM
done
```

attention aux boucles infinies :

```
while :
do
        echo "Do something; hit [CTRL+C] to stop!"
done
```

**Exemple d'utilisation normale de case et while :**

```
while :
  do
  echo "1 : création d'un compte utilisateur"
  echo "2 : modification d'un compte utilisateur"
  echo "3 : suppression d'un compte utilisateur"
  echo "q : quitter
  read A
  case $A in
    1) créer-compte.sh ;;
    2) modif-compte.sh ;;
    2) suppr-compte.sh ;;
    q) exit 0 ;;
    *) echo "saisie incorrecte" ;read ;;
  esac
done
```

**until**

syntaxe : `until [ condition ]; do commands; done`

Boucle jusqu'à ce que la condition soit vraie. opposé de while

```
until [ –n "$NOM" ]
  do
  read –p "Entrez votre nom : " NOM
done
```

**for**

syntaxe : `for item in [LIST] ; do commands; done`

Test sur des valeurs inconnues ou sur une liste

```
for A in *.rpm
  do
  rpm –qpl $A|grep libtruc.so >/dev/null && echo $A
done
```

**IFS:**

IFS : champ de separation utilisé par « `for` ».

```
set | grep ^IFS
```

```
IFS=$' \t\n'       # Indique que le champ de séparation est tabulation ou espace ou nouvelle ligne
```

**Exemple de boucle for en utilisant une liste :**

```
for VAR in 1 2 3 Toto
do
echo VAR
done
```

**Exemple de boucle for en utilisant une boucle de valeur :**

```
for (( i = 10 ; i >= 1 ; i –= 1 ))      # valeur initiale, condition, incrementation/décrémentation
  do
  echo "mise à feu dans $i seconde(s)"
  sleep 1
  done
echo '!!! BOUM !!!'
```

# 5 Utilisation des fonctions

Une fonction peut être présente :

- dans le script qui l'exécute
- dans un autre fichier (dans ce cas on fait un include)

**Exemple de fonction presente dans le script qui l'exécute :**

```
Bienvenue() {                                   # création de la fonction
        echo "Bienvenue $LOGNAME"
        echo Nous sommes le `date +%D`
        }
Bienvenue                                       # appelle la fonction
```

La commande doit être lue par le Shell AVANT d'être appelée !!

**Arguments dans les fonctions :**

```
Bienvenue() {
        echo "Bienvenue $2 $1"                  # rappelle les arguments 2 et 1
        echo Nous sommes le `date +%D`
        }
read –p "quel est votre prénom : " PRENOM
read –p "quel est votre nom : " NOM
Bienvenue $PRENOM $NOM
```

**Externaliser les fonctions**

`vi fichier.fonction` : création du fichier de fonction

```
bienvenue() {
echo "Bienvenue $2 $1"
return                                          # fait quitter la fonction
echo "Nous sommes le `date +%D`"
}
```

`vi script.sh` : création du script principal

```
# !/bin/bash
. fichier.fonction                              # include du fichier
read –p "quel est votre nom : " nom
read –p "quel est votre prénom : " prenom
bienvenue $prenom $nom
```

### 5.1.1.1 A propos de « l'include » (appel de fonction)

`. fichier.fonction` : include en utilisant le chemin relatif de `fichier.fonction`.

On peut entrer le chemin absolu : `. /data/scripts/fichier.fonction`

# 6 Des commandes pour les scripts

**eval**

`eval` évalue la valeur de plusieurs arguments. Si aucune valeur n'est trouvée, retourne zéro.

```
vert="\033[32;1m"
rouge="\033[31;1m"
read -p "Choisissez une couleur (vert ou rouge) : " COULEUR
eval echo -e "\$$COULEUR Bonjour"                    # calcule $couleur puis calcule $vert ou $rouge
```

`echo -e "$$COULEUR Bonjour"` n'est pas correct car $$ renvoit le PID

**trap**

`trap` réaffecte les signaux envoyés au processus. Nota : on ne peut pas trapper `kill -9`

syntaxe : `trap [commande de remplacement] [signal]`

```
trap 'rm -f *.tmp ;exit1' INT TERM
```

**clear**

`clear` affiche un écran vide. Ne vide pas l'écran.

**script**

`script` enregistre tout ce qui apparaît à l'écran dans un fichier, pratique pour logger

**exemple :**

```
if test -t 0 ; then
  script
  exit
fi
```

**tr**

`tr` effectue du remplacement de caractère, de la suppression de doublons. Mode caractère

- `cat fonctions.sh | tr 'abc' 'yxz'` : remplace a par x, b par y et c par z
- `tr '[:lower :]' '[:upper:]'` : remplace les minuscules par des majuscules
- `tr -d 'abc'` : supprime les a, les b et les c
- `tr -s '\n'` : supprime les lignes vides
- `tr -s '\t' ' '` : remplace les `\t` (tabulations) par des espaces et remplace les doublons d'espaces

**cut**

`cut` découpe verticalement (caractère/champ)

- `cat /etc/passwd | cut -c 1,5-10` : récupère les caractères 1 et 5 à 10 (collés)
- `cut -f1 -d:` : récupère le 1er champ. Utilise « : » comme délimiteur

```
VAR='     olivier dehecq'
echo $VAR | tr -s ' ' '|' | cut -f1,4 -d'|'       # enlève les espace en trop, sélectionne les champs
```

**split**

`split` découpe les fichiers en blocs de lignes

`split gros-fichier.log petit-fic-` : par défaut découpe en morceaux de 100 lignes

```
petit-fic-aa
petit-fic-ab
petit-fic-ac …
```

inverse de split : `cat petit-fic?? > fusion_de_petit-fic.txt`

**sort**

`sort` permet de trier les colonnes

- `cat client | sort` : trie de A à Z du premier caractère jusqu'au dernier caractère
- `cat client | sort -k2` : trie de A à Z de la **deuxième** colonne jusqu'au dernier caractère
- `cat client | sort -k3n` : trie **de 0 à 9** de la troisième colonne jusqu'au dernier caractère
- `cat client | sort -k2r` : trie de **Z à A** de la 2e colonne jusqu'au dernier caractère
- `cat client | sort -k1,1 -k3n` : trie de A à Z le champ1 puis de A à Z du troisième champ à la fin
- `sort -b -k4.5 -k4.3,4.4 -k4.1,4.2` : `k4.1,4.2` : champ4 premier caractère et champ4 deuxième caractère
  nota : `-b` : ne pas tenir compte des espaces **avant**

**uniq**

`uniq` supprime les lignes consécutives en plusieurs exemplaires

les lignes doivent être **consécutives**, on fait un `sort` avant !

Syntaxe : `uniq [options] fichier1 [fichier2 …]`

- `-c` : chaque ligne est préfixée du nombre d'occurrences
- `-d` : n'affiche que les lignes en plusieurs exemplaires
- `-f3` : compare les 3 premiers champs sur la ligne.
- `-w3` : compare les 3 premiers caractères sur la ligne
- `-i` : ignore la casse

**Exemple :**

```
sort fic1 | uniq
```

# 7 La commande sed

Sed = stream editor. Pour manipuler du texte

`info sed` : **manuel très complet de sed**

`ls –l | sed 'p'` : imprime les lignes qui correspondent. Double les lignes car affiche les lignes qu'il a reçu en entrée.

`ls –l | sed –n 'p'` : fait la même chose mais n'affiche pas les lignes reçues en entrée.

`ls –l | sed –n '/mm/p'` : affiche uniquement les lignes contenant `mm` (revient à faire du `grep`)

`ls –l | sed –n '/^t/p'` : affiche uniquement les lignes **`commencant par t`** (prend du regex)

`ls –l | sed –n 's/^total/toto/'` : substitue (`s`) les lignes commençant par total et les remplace par toto

`ls –l | sed –r 's/.*:[0–9]{2}\s(.*)/\1/'` : pour chaque ligne, **récupère le contenu de () et affiche la uniquement première \1 occurrence correspondant à ce qui est entre parenthèse**

`–r` utiliser une expression régulière

`.*:[0–9]{2}\s.*` : `.*` n'importe quelle chaine de caractère suivie de `:` puis de `{2}` digits `[0–9]` puis de `.*`

**Substitution :**

`ls –l | sed –n 's/r/R/3'` : substitue la `3`e occurrence du r de chaque ligne par R

`ls –l | sed –n 's/r/R/g'` : substitue **`tous`** les r de chaque ligne par R

`ls –l | sed –n 'y/rw/RW/'` : substitue **`toutes les occurrence de chaque lettre par son 2e motif`** (équivalent à `tr`)

**numéro de ligne :**

`ls –l | sed –n '='` : indique le numéro de chaque ligne traitée

`ls –l | sed –n '$='` : indique le nombre de lignes ($ : la dernière ligne. Équivalent à `wc`)

`ls –l | sed –n '/mm/='` : indique le numéro de ligne correspondant à la chaine recherchée

**Ligne avant, après, supprimer :**

`ls –l | sed '/mm/aNew line'` : ajoute une nouvelle ligne New line **après** la ligne contenant mm

`ls –l | sed '/mm/iNew line'` : ajoute une nouvelle ligne New line **avant** la ligne contenant mm

`ls –l | sed '/mm/cReplace line'` : met la ligne Replace ligne **à la place de** la ligne contenant mm

`ls –l | sed '/mm/d'` : **supprime** la ligne contenant mm

`ls –l | sed '/mm/!d'` : fait **l'inverse de supprime** la ligne contenant mm. N'affiche que la ligne contenant mm

`ls –l | sed –n '2,3y/rw/RW/'` : n'effectue la fonction que sur les 2e et 3e lignes

`ls –l | sed –n '2,/mm/y/rw/RW/'` : n'effectue la fonction que sur la ligne 2 et la ligne contenant mm

**Utiliser un fichier de paramètres :**

exemple.sed

```
/total/d                                    # supprime les lignes contenant total
s/août/AOUT/                                # remplace la 1ere occurrence de août par AOUT
```

`ls –l | sed –f exemple.sed` : applique toutes les fonctions du fichier exemple.sed

`sed –i.bak 's/Aout/Août/' exemple.txt` : ecrase le fichier exemple.txt (mais crée un fichier exemple.txt.bak)

Il est aussi possible (mais dangereux) de faire : `sed –i 's/Aout/Août/' exemple.txt` ne crée pas de .bak

# 8 La commande awk

**Manipulation de champs :**

- Séparateur par défaut = tabulation ; `–F ";"` : définir ; comme séparateur
- `$1` : premier champ ; `$2` : deuxième champs … `$0` : tous les champs : `$NF` : dernier champ

`cat monfichier.txt | awk '{print $1"||"$2}'` : affiche et concatène : champ1||champ2

script.sh

```
#!/usr/bin/awk –f
BEGIN {
print "Bonjour toto,\nje suis ton script awk !"
}
```

`chmod +x script.sh ; ./script.sh` : rend le script exécutable puis exécute le script

`cat fichier.csv |awk –F ";" '{print $1}'` : séparateur ; et affiche le premier champ

**Numéro de champ NF, de lignes NR et pied de commande END :**

- `NF` : Field Number numéro de champ
- `NR` : Rows Number afficher un numero de ligne
- `END` : footer (pied de tableau)

`cat fichier.txt | awk '{print NR " => "$0}'` : affiche le numéro de chaque ligne

`cat fichier.txt | awk '{print NF " => "$0}'` : indique le nombre de champs

`cat fichier.txt | awk 'END {print NR}'` : en footer, nombre de lignes du tableau

`cat fichier.txt | awk '{print $0} END {print "Nb lignes: "NR}'` : on peut mixer

**Recherche de pattern et BEGIN :**

- `/pattern/` : rechercher un pattern
- `!` : inverser le pattern
- `BEGIN` : comme END mais en entête de tableau
- `Length()` : nombre de caractères

script.sh

```
cat myfile.txt | awk '
BEGIN {print "\nLigne des totoistes :\n"}
!/toto/ {print "Prenom: "$2" – Ville : "$3" – Département: "$4" – long_prenom = "length($2)}
END {print "\nLigne des totoistes :\n"}
'
```

Pour le pattern on peut aussi utiliser `!/xav[ie]+r` : contient xav puis i ou e répétés 1 ou plusieurs fois puis r

**Remplacement avec gsub() :**

- `gsub("pattern","remplacement","champs")` : syntaxe

`cat fichier.txt | awk '{gsub("ier","ki",$2);print$2}'` : remplacer ier par ki dans le champ2 puis afficher le champ2

**Condition IF :**

- `if(condition1) action else if(condition2) action2 else action3` : syntaxe

`cat fichier.txt | awk '{if(NR==1) print $1 ;else if ($2=="xavier") print $0;}'` : imprimer le premier champ si c'est la première ligne, sinon imprimer toute la ligne si le 2e champ égal « xavier »

Nota : équivalent à (mode ternaire) : `cat fichier.txt | awk '{print NR==1? $1:$2=="xavier"? $0:""}'`

**MATCH pour capturer une expression :**

- `match(field,/pattern/)` : syntaxe pour capturer un pattern
- `print substr($0, RSTART, RLENGTH)` : syntaxe pour afficher la capture à partir d'un découpage

exemple

`cat fichier.txt |`

`awk 'match($0,/192(.[0-9]+)+/) {print substr($0, RSTART, RLENGTH)}'` : imprimer les addresses IP

**Analyse des logs Apache :**

Exemple de fichier /var/log/apache2/access.log :

```
127.0.0.1 - - [08/Dev/2020:20:22:20 +0100] "GET / HTTP/1.0" 200 11192 "-" "ApacheBench/2.3"
...
```

Script :

```
cat /var/log/apache2/access.log | awk '
{
match($4,/\[(.*):/)                # récupère dans la colonne 4 celle qui contiennent des dates
gsub(":.*|\[","",$4)               # remplacer ce qui commence par :.* ou les [ et remplacer par ""
tab[$4" - "$1]++                   # crée un tableau $4 - $1 et à chaque fois incrémente à chaque
fois
}
END
{
for (i in tab)                     # pour chaque dernière occurrence de tab
print i" - hits: "tab[i]           # afficher les infos : i = ligne / tab[i] = valeur auto-
incrémentée
}'
```

**Cumul et définition de variables :**

- Dans awk, une variable ne prend pas de $
- Les variables sont notamment pratiques pour faire des cumuls car on peut les incrément et faire un print dans END

Script pour afficher le nombre de mots (séparés par des espaces) :

```
cat fichier.txt | awk '
{total = total + NF;}
END
{print "Nb mots: " total}'
```

Script pour afficher le nombre de caractères (séparés par des espaces) :

```
cat fichier.txt | awk -F "" '
{gsub(/\t/,"",$0); total = total+NF;}         # /pattern/ : en regex \t = tabulation
END
{print "Nb cara: " total}'
```

**Split et for :**

- Dans awk, une variable ne prend pas de $
- `tab[index]` : tableau/array, ne prend pas de $ ; `tab` = nom du tableau / `index` = valeur de l'index
- `for(i=1 ;i<=10 ;i++){print i}` : la variable `i` 10 fois

`cat monfichier.txt | awk '{ split($0,tab, " "); print tab[2]}'` : imprimer les 2e mot de chaque ligne

script :

```
cat monfichier.txt | awk '{
split($0,tab, "");                      # séparer chaque caractère dans un tableau
for (i=1 ;i<=lenght($0) ;i++)           # nombre de champs dans la ligne
print tab[i]}'                          # affiche chaque caractère sur une ligne
```

# 9 Plus loin avec les variables

**Substitution des variables**

`echo ${nom – invité}` : affiche $nom. Si la variable n'existe pas, affiche « invité »

`echo ${nom :– invité}` : affiche $nom. Si n'existe pas ou valeur nulle affiche « invité » (utile après un read)

`echo ${nom := invité}` : affiche $nom. Si n'existe pas ou null, définit la variable nom="invité"

`while ${#tel != 10}` : nombre de caractères de $tel, teste si différent de 10

`var=fic.save;echo ${var#??}` : retire de la gauche vers la droite 2 caractères quelconques

`echo ${var#.*}` : retire de la gauche vers la droite la première occurrence trouvée de `.*`

`echo ${var##.*}` : retire de la gauche vers la droite la dernière occurrence trouvée de `.*`

`echo ${var%/*}` : retire de la droite vers la gauche la première occurrence trouvée de `/*`

`echo ${var%%/*}` : retire de la droite vers la gauche la dernière occurrence trouvée de `/*`

**L'arithmétique entière**

`let` **et** `(( ... ))` **:**

`typeset –i var=10` : déclare le typeset (type de variable) de var comme étant un entier (integer) et égal à 10

`unset var=toto` : enlève le typeset

`let var=5+10` : calcule et attribue à var la valeur 15

`let nb=var*2` : calcule et attribue à nb la valeur 30

`(( nb = var * 2 ))` : fait la même chose que `(( nb=$var*2 ))`

**Incrémenter :**

`(( nb=nb+1 ))` est identique à `(( nb+=1 ))`

Exemple :

```
min=100
(( heure=min/60 ))
(( minutes=min%60 ))
```

Autre exemple :

```
min=100
echo $(( heures=min/60 ))H$(( minutes=min%60 ))min                    # % veut dire modulo
```

```
1H40min
```

**Comparer :**

<, > : inférieur à, supérieur à

<=, >= : inférieur ou égal, supérieur ou égal

!=, == : différent de, égal à

&&, || : et (logique), ou (logique)

! : n'est pas (inverse)

Exemple :

`(( k==i+j && h==k*2 )) ; echo $?` : affiche le code retour de `k égal i+j et h égal k*2`

**Les variables de tableau**

`client=Dupont` : la valeur de la variable client vaut `Dupont`

`echo $client` : affiche `Dupont`

`echo ${client}` : affiche `Dupont`

`echo ${client[0]}` : affiche `Dupont` (Dupont étant la valeur de l'index 0 de la variable de tableau client

On peut remplacer par `[1]` : deuxième valeur. On peut remplacer par `[*]` : valeurs de tous les index

`client[1]=Alexandre` : la valeur de l'index 1 de la variable de tableau client vaut `Alexandre`

`echo ${client[0]}` : affiche `Dupont`

`echo ${client[1]}` : affiche `Alexandre`

`echo ${client[*]}` : affiche `Dupont Alexandre`

**Exemple :**

```
client="Dupont"
client[1]=Alexandre
nom=0 ; prenom=1
echo ${client[prenom]}                          # affiche Alexandre
```

`echo ${#client[*]}` : affiche le nombres d'index utilisés dans la variable de tableau